On Combining Formal and Informal Verification

Jun Yuan* Jian Shen** Jacob Abraham** Adnan Aziz**

Abstract. We propose algorithms which combine simulation with symbolic methods for the verification of invariants. The motivation is twofold. First, there are designs which are too complex to be formally verified using symbolic methods; however the use of symbolic techniques in conjunction with traditional simulation results in better "coverage" relative to the computational resources used. Additionally, even on designs which can be symbolically verified, the use of a hybrid methodology often detects the presence of bugs faster than either formal verification or simulation.

1 Introduction

In this paper we will be concerned with the problem of *design verification*; specifically, the problem of *invariant checking* over gate-level designs. Traditionally, designs have been verified by extensive simulation. While offering the benefits of simplicity and scalability, simulation offers no guarantees of correctness; for large designs, the fraction of the design space which can be covered in this methodology becomes vanishingly small. Indeed, there are many examples of designs that passed extensive simulation, but were still found to contain bugs [4]. This has led to the proposal of "formal methods" for design verification; the adjective formal refers to the unambiguous specification of the system and the properties being checked, together with the validation step generating a mathematically rigorous proof of correctness.

In theory the computational complexity of invariant checking on netlists is high; PSPACE-complete to be precise. In practise, many designs are well structured, and this can be exploited to devise heuristic procedures which perform well on specific classes of designs. One method which has been used to successfully verify a large number of complex designs is the use of "symbolic data structures" such as Binary Decision Diagrams (BDDs) to efficiently represent and manipulate the state spaces of designs [11]. The primary limitation of BDD based approaches to invariant checking is that for many designs, the BDDs constructed in the course of verification can grow extremely large, resulting in space-outs or severe performance degradation due to paging [12].

Practicing verifiers are less concerned with formally verifying designs than finding bugs in them as early as possible. As Henzinger has pointed out, "falsification" is a more accurate description of the endeavor called "verification" Faced

* Motorola Inc., Austin TX yuan@adttx.sps.mot.com
** ECE Dept., Univ. of Texas, Austin TX {jshen | jaa | adnan} @ece.utexas.edu

with the twin dilemmas of diminished coverage through simulation and the inability of symbolic methods to formally verify large designs, it is natural to ask how best to combine symbolic methods with simulation, so as to find bugs as quickly as possible.

In this paper we provide two answers to the problem posed above. It is to be stressed that neither method is complete, i.e., guaranteed to provide a formal certificate of correctness if the invariant passes, or a counterexample if it fails. However, all reported violations of the invariant are true bugs.

We first develop the method of *saturated simulation*, wherein the designer designates a subset of the latches as being "interesting"; these could, for example, be the program counter and status bits in a microprocessor. The procedure performs a partial traversal of the state space. At each step, symbolic techniques are used to compute the image of current set of states, and only a minimal subset of the image is kept so that all control states seen thus far are represented; we also describe an extension that visits all controller edges. Heuristically, the control portion of the design, while being much smaller than the datapath, is the main source of design errors. Saturated simulation attempts to explore as much of the control state space, thus increasing the likelihood of finding bugs.

The efficiency of this approach comes from the observation that it is feasible to compute the symbolic image of a single state even for very large designs, coupled with the fact that the set of control states is typically much smaller than the entire state space. Additionally, fast BDD routines exist for generating and manipulating representative elements of equivalence classes [9].

We then describe an orthogonal approach referred to as *retrograde analysis* [15]. Starting from B_0, the complement of the invariant, successive preimages $B_0, B_1, B_2, \ldots$ are computed symbolically. This is done till the BDD for some B_n grows larger in size than a (user-specified) threshold value. Cycle simulation is performed from an initial state; simulation is halted if a state which lies B_n, since every state in B_n can reach a state in B_0. We describe greedy search strategies for finding paths to B_n from an initial state which use Hamming distance as a metric to be minimized. The primary benefit of retrograde analysis is that the set $\cup_i B_i$ is typically much larger (in the sense of cardinality) than B_0; hence, in a heuristic sense, B_n offers a much larger "target" for simulation.

These routines have been coded on top of the tool VIS [2]. Our experimental results underline the effectiveness that is suggested by the heuristic arguments given above.

To the best of our knowledge, the principles of saturated simulation and retrograde analysis are novel to this paper. We have been influenced by a number of related works. Thompson's [15] work on Retrograde Analysis provided the initial impetus. Additionally, we were influenced by the dramatic improvements made to cycle simulation by the use of BDDs by Ashar and Malik [1], and McGeer et al. [10], who made clear the importance of making maximum use of the physical memory available on the machine. Ravi et al. [13] attempt to pick subsets of state sets encountered during reachability analysis which have small BDDs but contain a large number of states. This is distinct from our approach, wherein a subset is

chosen which attempts to maximize the number of distinct controller states. Cho et al. [5] pick nets to abstract into primary inputs, consequently obtaining supersets of the set of reachable states. The work of Ho et al. [7] and Hoskote et al. [8] on creating simulation vectors which excite a large number of transitions on the controller states of a design suggested the usefulness of using transitions rather than states to obtain good coverage of controller behavior. However, they used designer supplied "translation functions", or test-based techniques to generate simulation input sequences which excited as much of the control as possible; our approach is rooted in symbolic methods.

2 Background — Invariant Verification

In order to be able to analytically reason about hardware, we first need to develop mathematical models for digital systems. Singhal [14] gives a detailed exposition for computational models for hardware.

Hardware designs can be modeled at the *structural* level using *netlists*, or at the *behavioral* level using *finite state machines* (FSMs). A netlist consists of an interconnected set of primary inputs, gates, and latches. Each gate has an associated Boolean function. A finite state machine can be represented by an edge-labeled directed graph, where the vertices correspond to *states*, and the labels are *input-output* pairs.

For a given a netlist η, there is a natural way of deriving a finite state machine from it; states are evaluations to the set of latch variables, and the next-state/output functions are derived by composing the gate functions.

Given a design D and a set of states A, the *image* of A (denoted by $Img(A)$) is the set of all states which can be reached from A by applying an input sequence of length one. Similarly, the *pre-image* of A (denoted by $PreImg(A)$) is the set off all states which can reach A in one step. The Img and Pre-image procedures can be implemented symbolically using Reduced Ordered Binary Decision Diagram [3].

A common verification problem for hardware designs is to determine if every state reachable from a designated set of initial states lies within a specified set of "good states" (referred to as the *invariant*). This problem is variously known as *invariant verification*, or *assertion checking*.

One straightforward solution to the invariant checking problem is to symbolically compute all states reachable from the initial states and determine that they all lie in the invariant. An alternate approach to checking invariants is based on *backward analysis*, wherein the symbolic *PreImg* operator is iteratively applied to determine all states which can reach the complement of the invariant; the invariant fails if the initial state lies in this set.

The primary limitation of both approaches is that the BDDs encountered in the course of image computations can grow very large.

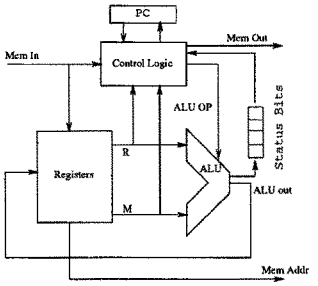


Fig. 1. Partitioning a design into Control and Datapath.

3 Saturated Simulation

Many designs can be separated into "control" and "datapath" as illustrated in Figure 1; furthermore, the designer is aware of this dichotomy. For most such designs, the number of latches in the controller is usually a small fraction of the total number of latches; however, the control portion is where bugs usually occur. In this section, we describe an approach we refer to as "saturate simulation"; this approach attempts heuristically to explore as much of the control portion of the design as possible.

As an example, consider the `viper` microprocessor [6]. It contains 9 latches which can be naturally designated control and 210 which are data. Hence, there are no more than 512 different possible values for the control state. It is feasible even for very large designs to compute the image of a small (in the sense of cardinality) set of states. In part, this follows from the fact that the construction of the BDD for the next-state logic can be restricted to the current set of states. This suggests that it may be possible to perform a "partial" reachability analysis, in which all distinct control states are preserved at each step.

Let the variables associated with the control portion of the design be X_c and the variables associated with the datapath be X_d. Thus the state of the design is given by an evaluation to $X_c \cup X_d$.

Definition 1. Let A be a set of states. A subset A' of A is *control-saturated* with respect to A if

$$(\forall \alpha_c . \forall \alpha_d)\,[\,(\alpha_c, \alpha_d) \in A \;\rightarrow\; (\exists \alpha_d')[(\alpha_c, \alpha_d') \in A']\,]$$

Intuitively, A' is a control-saturated subset of A if every control state occurring in A occurs in A'. Thus control-saturated subsets of A preserve all the controller states present in A. Heuristically, a minimal control-saturated subset of A is a good representative set — it includes all the distinct controller configurations in A, and is as small as possible (in the sense of cardinality). An example of a control-saturated subset is given in Figure 2(a).

We now address the problem of computing minimal control-saturated subsets of A. Let f be a Boolean function on variables $X = \{x_1, x_2, \ldots, x_n\}$. Lin et

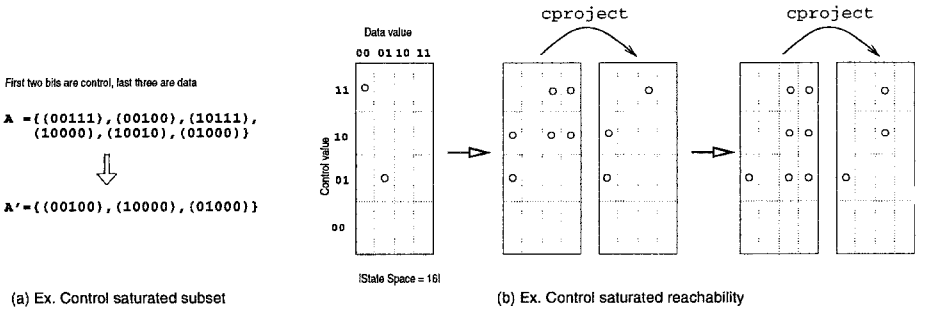


Fig. 2. Minimal control saturated subsets.

```
 /* A initialized to the BDD for the reset states. */
 /* G is the BDD for the invariant.  */
BDD_t function Cntrl_Sat_Sim(A, Cntrl_Vars, G) {

  if (BDD_Intersects(A, Ḡ))  /* Invariant fails!! */
     assert FAIL;

  ImgA := BDD_Img(A);
  R := BDD_Or( A, ImgA );
  R* := BDD_Cproject( R, Cntrl_Vars );

  if (BDD_Equal( R*, A))
     return R*;

  return Cntrl_Sat_Sim(R*, Cntrl_Vars, G);
}
```

Fig. 3. Control-saturated simulation.

al. [9] presented an efficient algorithm (referred to as the `cproject` operator) which takes a BDD for f and a subset $X' \subset X$ of the variables, and returns a BDD for a function f^* which has the property that

1. for any assignment v to the variables in X, so that $f(v) = 1$, there is exactly one valuation v' which agrees with the valuation v over the variables in X' so that $f^*(v') = 1$, and furthermore
2. for all u, $f^*(u) = 1 \Rightarrow f(u) = 1$.

Since sets can be thought of in terms of characteristic functions, we will freely apply the `cproject` operator to sets. Observe that `cproject`(A, X_c) is a minimal control saturated subset of A.

In Figure 3 we sketch a simple symbolic procedure for invariant verification. Reachable states are iteratively computed using the *Img* operator; at each step, a control-saturated subset of the current reached state is computed using the `cproject` operator. This in turn is used as the current reached state set. The first

few steps are illustrated in Figure 2(b). The procedure is incomplete, since it is greedy: minimal control-saturated subsets of the sets computed by the `cproject` operator will not necessarily be sufficient to cover all possible controller states.

One simple way of further enhancing the coverage achieved by control-saturated simulation is to generate several "representative" control states. There are simple modifications to the `cproject` operator which can achieve this effect. Another approach is to apply `cproject` only to the frontier of the reached states at each iteration.

3.1 Control-edge Saturated Simulation

A fundamental extension to obtain enhanced coverage is to perform a partial reachability analysis and at each step pick a subset of the image which preserves all "controller transitions" to the image from the current set. Ho et al. [7] and Abraham et al. [8] created simulation vectors which excite a large number of control transitions in designs; the high quality of their results in terms of finding bugs with these vectors underlines the usefulness of using transitions rather than states to obtain good coverage. As an example, consider a microprocessor where the control state is the value of the program counter. Two states which correspond to different lines in the program may both transition the same program line with different data values; in this case, it is natural to keep the resulting states different.

We now describe how to explore edges in the control state space.

Definition 2. Let A be a set of states. A subset B of $Img(A)$ is said to be *control-edge saturated* with respect to A if

$$(\forall\alpha_c.\forall\alpha_d.\forall\beta_c.\forall\beta_d)\big[\,[(\alpha_c,\alpha_d)\in A\ \wedge\ (\beta_c,\beta_d)\in Img(\{(\alpha_c,\alpha_d)\})]\ \rightarrow$$
$$(\exists\beta_d{}'.\exists\alpha_d{}')[(\beta_c,\beta_d{}')\in B\ \wedge\ (\alpha_c,\alpha_d{}')\in A\ \wedge\ (\beta_c,\beta_d{}')\in Img(\{(\alpha_c,\alpha_d{}')\})]\,\big]$$

In English, the above definition says that B is control-edge saturated when for every transition $(\alpha_C,\alpha_D)\rightarrow(\beta_C,\beta_D)$ from A to $ImgA$, there is a state $(\beta_C,\beta_D{}')$ in B and a state $(\alpha_C,\alpha_D{}')$ in A so that $(\alpha_C,\alpha_D{}')\rightarrow(\beta_C,\beta_D{}')$.

Thus in some sense, control-edge saturated subsets of $Img(A)$ preserve all the controller transitions originating at A. Heuristically, a minimal control-edge saturated subset of $Img(A)$ is a good representative set — it includes all the distinct controller configurations resulting in $Img(A)$ from transitions from A, and is as small as possible. An example of a minimal control-edge saturated subset is given in Figure 4.

Minimal control-edge saturated sets can be computed augmenting the design: for every control latch x_C, add a new latch x_S which "shadows" x_C, that is, the next state of x_S is the present state of x_C. Denote the set of shadow state variables thus introduced by X_s. Clearly the next-state of the latches indexed by $X_c\cup X_d$ is independent of that of the shadow latches. The following lemma demonstrates that minimal control-edge saturated sets can be computed from the augmented design.

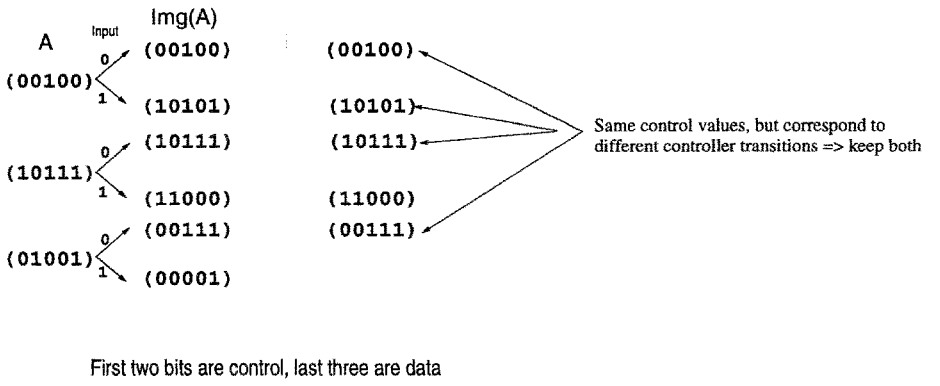


Fig. 4. A minimal control-edge saturated subset.

Lemma 3. Let A^* be A lifted from $X_c \cup X_d$ to $X_c \cup X_d \cup X_s$. Define B to be the existential quantification of $\texttt{cproject}(Img(A^*), X_c \cup X_s)$ by X_s. Then B is minimal control-edge saturated with respect to A.

Proof. Observing that $\texttt{cproject}(\Gamma, \theta)$ is always subset of Γ, it follows that $\texttt{cproject}(Img(A^*), X_c \cup X_s)$ is a subset of $Img(A^*)$. Since the next state of non-shadow latches does not depend on the shadow latches, it follows that the existential quantification of $Img(A^*)$ by X_s is equal to $Img(A)$, and so B is a subset of $Img(A)$.

We now show B is control-edge saturated with respect to A. Let (α_C, α_D) and (β_C, β_D) satisfy the "if" portion of the implication in Definition 2. Then there is a transition from $(\alpha_C, \alpha_D) \in A$ to (β_C, β_D), i.e., $(\beta_C, \beta_D) \in Img(\{(\alpha_C, \alpha_D)\})$. ¿From the construction of the augmented design, $((\beta_C, \alpha_C), \beta_D)$ is in $Img((\alpha_C, \alpha_S), \alpha_D)$ for an arbitrary assignment α_S to the shadow latches. Hence $\texttt{cproject}(Img(A^*), X_c \cup X_s)$ contains a state of the form $((\beta_C, \alpha_C), \beta_D')$. Note $((\beta_C, \alpha_C), \beta_D{}')$ lies in $Img(A^*)$; let it lie in the image of $((\alpha_C, \alpha_S'), \alpha_D{}')$. Hence, on existentially quantifying the X_s variables from $\texttt{cproject}(Img(A^*), X_c \cup X_s)$, the resulting set (namely B) will contain (β_C, β_D'). Since $(\beta_C, \beta_D{}')$ lies in the image of $(\alpha_C, \alpha_D{}')$, $\alpha_D{}'$ and $\beta_D{}'$ are existential witnesses for the "then" portion of the implication in Definition 2.

Minimality of B follows from the properties of `cproject` described in the previous section. ∎

3.2 Experimental Results – Saturated Simulation

We coded the routines described in the previous section as part of the VIS program [2]. Results are provided on two benchmarks – the *8085*, and *viper* microprocessors. The *8085* is approximately 4000 gate equivalents, and contains 242 latches, of which 33 were identified as being control. The *viper* is also 4000 gate equivalents, and contains 218 latches of which 9 were from the control. All experiments were conducted on an UltraSPARC 1, with a 170 MHz processor, and

Example	*Rchd. States*	Peak BDD	*Control States*	*Control Edges*	depth
viper	1.36×10^{19}	2033	23	31	4
8085	1.43×10^{7}	275641	1233	3723	10

Table 1. Complete BDD based reachability analysis.

Example	Peak BDD	*Control States*	*Control Edges*	depth
viper	160180	246	688	64
8085	81089	1846	4765	43

Table 2. Partial reachability analysis using control-state saturated subsets.

128 MBytes of main memory. A timeout of 2000 seconds was used for all *viper* experiments, and 1000 seconds for *8085* experiments. Sifting-based dynamic reordering was enabled throughout the experiments.

Table 1 presents results on the use of a complete BDD-based reachability analysis on the two benchmarks. Peak BDD is the number of nodes in the largest BDD encountered during reachability analysis. (The abnormally low peak BDD for *viper* in Table 1 stems from the fact that the program timed out after the first four reachability steps, which were easily performed.) Table 2 presents results on the use of a control-state saturated simulation (as given in Figure 3). For *8085*, we compute almost twice as many reachable control states and transitions; for *viper*, an order of magnitude more. Table 3 presents results on the use of control-edge saturated simulation. In the same time, more edges are visited; this comes at the expense of higher memory consumption with respect to control-state saturated simulation. Interestingly, fewer control states are visited; we ascribe this to the fact that the control-state saturated simulation is faster, and so manages to go deeper into the state space in the same amount of time; this is seen in the depth column.

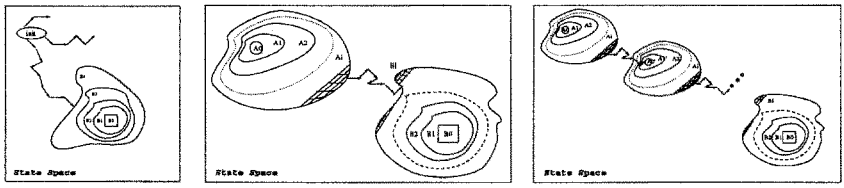
We compare saturated simulation with fast lookup based cycle simulation [1, 10] in Table 4. For *viper*, we performed 1000 sets of simulations, each comprising of 200 vectors; for *8085* we performed 4000 sets of length 200. Even though we gave cycle simulation two orders of magnitude more time, it still performed far worse than saturated simulation.

Example	Peak BDD	*Control States*	*Control Edges*	depth
viper	71213	236	705	60
8085	81089	1696	6324	30

Table 3. Partial reachability analysis using control-edge saturated subsets.

Example	Saturated Simulation			Cycle Simulation			
	Time (sec)	|Ctl States|	|Ctl Edges|	Time	Size	|Ctl States|	|Ctl Edges|
viper	2000	236	705	86616	1000 × 200	121	288
8085	1000	1696	6324	99143	4000 × 200	705	2674

Table 4. Comparing saturated simulation with cycle simulation.



(a) Vanilla RA (b) RA with Hamming; closest states are hatched (c) Enhanced RA with Hamming

Fig. 5. Retrograde search for Invariant checking

4 Retrograde Analysis

Retrograde Analysis (RA) is an important search technique developed within the field of Artificial Intelligence. In its simplest form, RA first marks all end positions (e.g., checkmate), and then by making *unmoves* from the end positions works its way back to the positions farthest from the end position, on the way determining the game theoretic values of *all* positions in the search space.

RA can naturally be applied to invariant checking: construct the sets of states $B_0, B_1, \ldots$ where B_0 is the complement of the invariant and $B_{i+1} = PreImg(B_i)$. Analogously to the W_i's above, the B_i's are effectively bad states. The B_0's can grow very large in terms of cardinality; it is natural to use BDDs to represent them compactly. Finally, when main memory is nearly exhausted, say at the set B_l, search for an input sequence which takes an initial state to a state in B_l.

The simplest search strategy is the simulation of random input vectors starting from a random initial state; the search halts if some state reached in this fashion lies in B_l. This approach is illustrated in Figure 5(a). Note that checking if a state lies in the set defined by a BDD is very fast — it take time proportional to the number of bits in the state, and is independent of the size of the BDD.

A more sophisticated search strategy is to pick an initial state which is "close" to the target states, i.e., to B_l. We propose the use of Hamming distance as a measure of closeness.

Recall that the Hamming distance between $\alpha, \beta \in \{0,1\}^n$ (denoted by $\Delta(\alpha, \beta)$) is the number of positions in which the α and β vectors differ. Consider the relations $H_0, H_1, H_2, \ldots, H_n \subset \{0,1\}^{2n}$ where $(\alpha, \beta) \in H_k$ iff $\Leftrightarrow \Delta(\alpha, \beta) \leq k$. The relation H_1 can be constructed directly using BDDs. The relation H_{i+1} satisfies

the following identity:

$$H_{i+1} = H_i \cup (\exists\gamma)[(\alpha,\gamma) \in H_i \wedge (\beta,\gamma) \in H_1]$$

Hence, the BDDs for $H_0, H_1, H_2, \ldots, H_n \subset \{0,1\}^{2n}$ can be easily constructed; furthermore a simple argument based on counting cofactors shows that they are small for the interleaved variable ordering.

The search for states in B_l can be enhanced by by first performing forward reachability from the initial states till the BDD for reached states reaches a threshold size. ¿From the outermost ring, pick a state (say α) which is closest to B_l, and then perform random cycle simulation from α. This is illustrated in Figure 5(b). Instead of cycle simulation from α, a combination of symbolic forward reachability analysis coupled with the the Hamming heuristic can be recursively applied. This illustrated in Figure 5(c).

4.1 Experimental Results – Retrograde Analysis

We coded the routines described in the previous section as part of the VIS program [2], and experimented with a number of examples. Representative results are provided on two benchmarks – *Mesh4* is a routing algorithm on a 4 by 4 mesh of nodes, and *Cube4* is hypercube based routing protocol. For both examples, we chose an invariant which fails.

Results on *Mesh4* are reported in Figure 6. We plot BDD size and cardinality after successive pre-images in Figure 6(a); both grow quickly. In Figure 6(b) we plot the number of simulation trials needed to reach a pre-image, starting from the initial state against the number of preimage steps taken; each trial consists of applying 100 random vectors. It is clear from the picture that this number decreases rapidly.

The effect of Hamming distance is given in Figure 7 for the *Cube4* example. Figures 7(a) and 7(b) are as before. In Figure 7(c), we show the effect of taking one forward step, and then picking a state in the image which is close to the target as opposed to a random state in the image; in Figure 7(d) we take two forward steps, and then pick a state which is close to the target. In both cases, there is an appreciable decrease in the number of simulation trials needed when Hamming distance is used. Interestingly, when a state in the image is picked at random, the performance is actually worse that simply starting at the initial state.

5 Conclusion

We investigated ways in which to combine symbolic verification with simulation. Specifically, we gave heuristic justification for saturated simulation and retrograde analysis. Experimental evidence corroborates that these approaches yield enhanced coverage and robustness. Thus the combination of formal and informal verification offers benefits not available in each independently.

In the future we intend to build upon the theme of relating formal and informal methods, particularly the problem of validating software for embedded controllers.

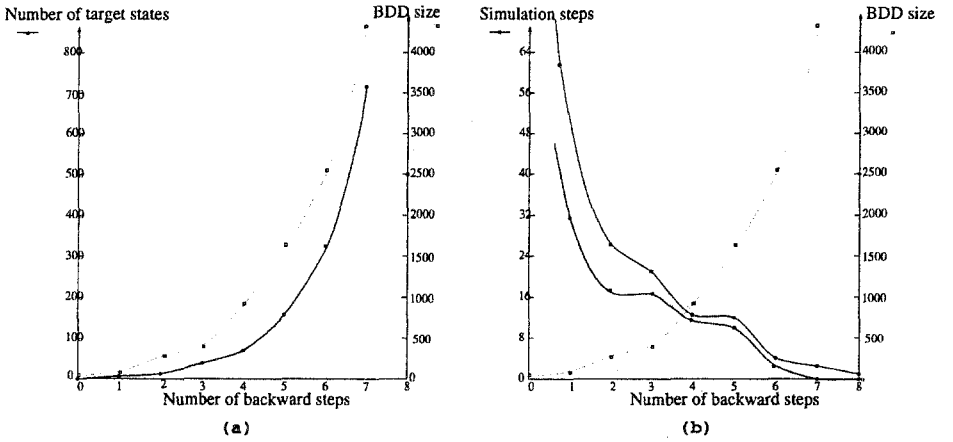


Fig. 6. Retrograde Analysis applied to *Mesh4*

References

1. P. Ashar and S. Malik. Fast Functional Simulation Using Branching Programs. In *Proc. Intl. Conf. on Computer-Aided Design*, November 1995.
2. R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. K. Ranjan, S. Sarwary, T. R. Shiple, G. Swamy, and T. Villa. VIS: A system for Verification and Synthesis. In *Proc. of the Computer Aided Verification Conf.*, July 1996.
3. R. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35:677–691, August 1986.
4. B. Chen, M. Yamazaki, and M. Fujita. Bug Identification of a Real Chip Design by Symbolic Model Checking. In *Proc. European Conf. on Design Automation*, pages 132–136, March 1994.
5. H. Cho, G. D. Hachtel, E. Macii, M. Poncino, and F. Somenzi. A Structural Approach to State Space Decomposition for Approximate Reachability Analysis. In *Proc. Intl. Conf. on Computer Design*, October 1994.
6. W.J. Culler. *Implementing Safety Critical Systems: The VIPER microprocessor.* Kluwer Academic Publishment, 1987.
7. Richard C. Ho, C. Han Yang, Mark A. Horowitz, and David L. Dill. Architectural Validation for Processors. In *Proceedings of the International Symposium on Computer Architecture*, June 1995.
8. Y. Hoskote, D. Moundanos, and J. Abraham. Automatic Extraction of the Control Flow Machine and Application to Evaluating Coverage of Verification Vectors. In *Proc. Intl. Conf. on Computer Design*, Austin, TX, October 1995.
9. B. Lin and R. Newton. Implicit Manipulation of Equivalence Classes Using Binary Decision Diagrams. In *Proc. Intl. Conf. on Computer Design*, Cambridge, MA, October 1991.
10. P. McGeer, K. McMillan, A. Saldanha, A. Sangiovanni-Vincentelli, and P. Scaglia. Fast Discrete Function Evaluation. In *Proc. Intl. Conf. on Computer-Aided Design*, November 1995.

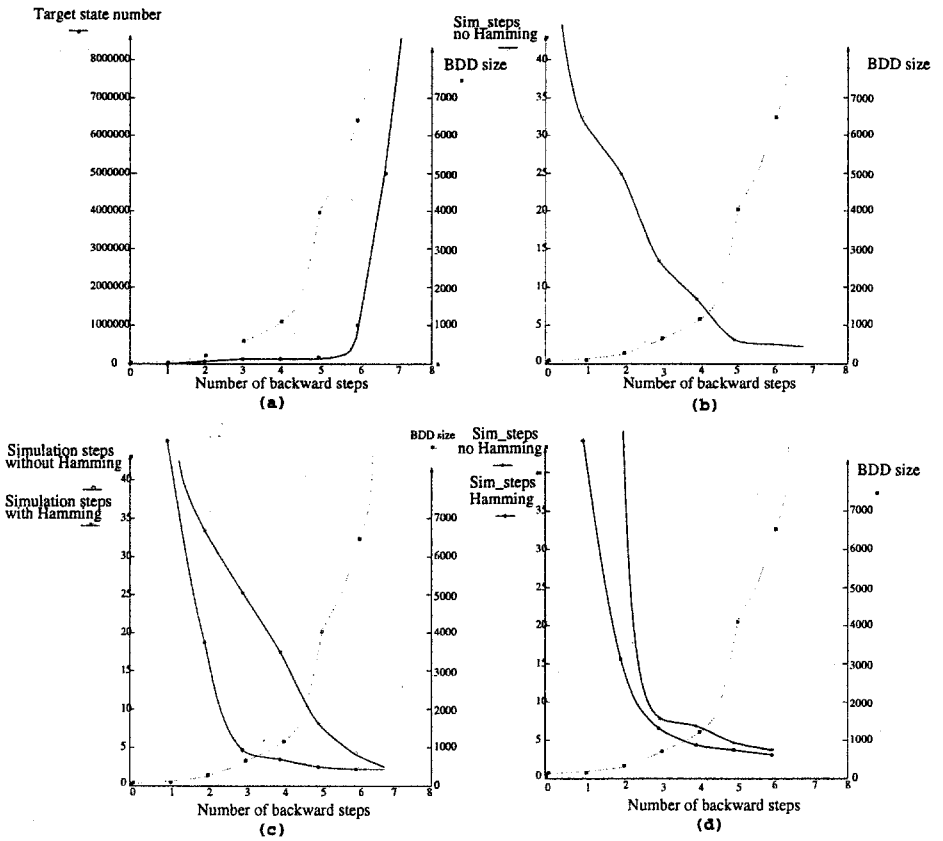


Fig. 7. Effect of Hamming Distance on *Cube4*

11. Kenneth L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.
12. R. Ranjan, J. Sanghavi, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. High Performance BDD Package Based on Exploiting Memory Hierarchy. In *Proc. of the Design Automation Conf.*, Las Vegas, NV, June 1996.
13. K. Ravi and F. Somenzi. High Density Reachability Analysis. In *Proc. Intl. Conf. on Computer-Aided Design*, Santa Clara, CA, November 1995.
14. Vigyan Singhal. *Design Replacements for Sequential Circuits.* PhD thesis, University of California Berkeley, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, 1996.
15. K. Thompson. Retrograde analysis of certain endgames. *ICCA Journal*, 9(3):131–139, 1986.